LAPORAN PRAKTIKUM 2
PEMROGRAMAN LANJUT
EXCEPTIONS DAN ASSERTIONS

Oleh

Nama	:	Rizqillah
NIM	:	1957301020
Kelas	:	TI 2C
Dosen Pembimbing	:	Musta'inul Abdi, SST., M.Kom.

PROGRAM STUDI TEKNIK INFORMATIKA

JURUSAN TEKNOLOGI INFORMASI DAN KOMPUTER

TAHUN 2021

LEMBAR PENGESAHAN

No. Praktikum	:	02/PPL/2C/TI/2021
Judul	:	Exceptions dan Assertions
Nama	:	Rizqillah
NIM / Kelas	:	1957301020 / TI 2C
Jurusan	:	Teknologi Informasi Dan Komputer
Prodi	:	Teknik Informatika
Tanggal praktikum	:	04 Maret 2021
Tanggal penyerahan	:	07 Maret 2021
Nilai	:	

Buketrata, 07 Maret 2021

Dosen Pembimbing,

Musta'inul Abdi, SST., M.Kom.
NIP. 19911030 20190310 1 5

DAFTAR ISI

BAB 1
PENDAHULUAN

1.1 Tujuan

Dasar penanganan exception telah dikenalkan pada anda di kursus pemrograman pertama. Bab ini membahas secara lebih dalam mengenai exception dan juga sedikit menyinggung tentang assertion.

Setelah menyelesaikan pembahasan, anda diharapkan dapat :

1. Menangani exception dengan menggunakan try, catch dan finally
2. Membedakan penggunaan antara throw dengan throws
3. Menggunakan exception class yang berbeda – beda
4. Membedakan antara checked exceptions dan unchecked exceptions
5. Membuat exception class tersendiri
6. Menjelaskan keunggulan penggunaan assertions
7. Menggunakan assertions

1.2 Dasar Teori

1.2.1 Apa itu Exception?

Bugs dan error dalam sebuah program sangat sering muncul meskipun program tersebut dibuat oleh programmer berkemampuan tinggi. Untuk menghindari pemborosan waktu pada proses error-checking, Java menyediakan mekanisme penanganan exception.

Exception adalah singkatan dari Exceptional Events. Kesalahan (errors) yang terjadi saat runtime, menyebabkan gangguan pada alur eksekusi program. Terdapat beberapa tipe error yang dapat muncul. Sebagai contoh adalah error pembagian 0, mengakses elemen di luar jangkauan sebuah array, input yang tidak benar dan membuka file yang tidak ada.

Seluruh exceptions adalah subclasses, baik secara langsung maupun tidak langsung, dari sebuah root class Throwable. Kemudian, dalam class ini terdapat dua kategori umum : Error class dan Exception class.

Exception class menunjukkan kondisi yang dapat diterima oleh user program. Umumnya hal tersebut disebabkan oleh beberapa kesalahan pada kode program. Contoh dari exceptions adalah pembagian oleh 0 dan error di luar jangkauan array.

Error class digunakan oleh Java run-time untuk menangani error yang muncul pada saat dijalankan. Secara umum hal ini di luar control user karena kemunculannya disebabkan oleh run-time environment. Sebagai contoh adalah out of memory dan harddisk crash.

Sebuah Contoh :

```
class DivByZero {
      public static void main(String args[]) {
            System.out.println(3/0);
            System.out.println(“Cetak.”);
      }
}
```

Jika kode tersebut dijalankan, akan didapatkan pesan kesalahan sebagai berikut :

```
Exception in thread "main" java.lang.ArithmeticException: / by
zero at DivByZero.main(DivByZero.java:3)
```

Pesan tersebut menginformasikan tipe exception yang terjadi pada baris dimana exception itu berasal. Inilah aksi default yang terjadi bila terjadi exception yang tidak tertangani. Jika tidak terdapat kode yang menangani exception yang terjadi, aksi default akan bekerja otomatis. Aksi tersebut pertama-tama akan menampilkan deskripsi exception yang terjadi. Kemudian akan ditampilkan stack trace yang mengindentifikasi method dimana exception terjadi. Pada bagian akhir, aksi default tersebut akan menghentikan program secara paksa.

Bagaimana jika anda ingin melakukan penanganan atas exception dengan cara yang berbeda? Untungnya, bahasa pemrograman Java memiliki 3 keywords penting dalam penanganan exception, yaitu try, catch dan finally.

Checked dan Unchecked Exceptions

Exception terdiri atas checked dan unchecked exceptions.

Checked exceptions adalah exception yang diperiksa oleh Java compiler. Compiler memeriksa keseluruhan program apakah menangkap atau mendaftar exception yang terjadi dalam sintax throws. Apabila checked exception tidak didaftar ataupun ditangkap, maka compiler error akan ditampilkan.

Tidak seperti checked exceptions, unchecked exceptions tidak berupa compile-time checking dalam penanganan exceptions. Fondasi dasar dari unchecked exception classes adalah Error, RuntimeException dan subclass-nya.

1.2.2 Apa itu Assertion?

Assertions mengijinkan programmer untuk menentukan asumsi yang dihadapi. Sebagai contoh, sebuah tanggal dengan area bulan tidak berada antara 1 hingga 12 dapat diputuskan bahwa data tersebut tidak valid. Programmer dapat menentukan bulan harus berada diantara area tersebut. Meskipun hal itu dimungkinkan untuk menggunakan contructor lain untuk mensimulasikan fungsi dari assertions, namun sulit untuk dilakukan karena fitur assertion dapat tidak digunakan. Hal yang menarik dari assertions adalah seorang user memiliki pilihan untuk digunakan atau tidak pada saat runtime.

Assertion dapat diartikan sebagai extensi atas komentar yang menginformasikan pembaca kode bahwa sebagian kondisi harus terpenuhi. Dengan menggunakan assertions, maka tidak perlu untuk membaca keseluruhan kode melalui setiap komentar untuk mencari asumsi yang dibuat dalam kode. Namun, menjalankan program tersebut akan memberitahu anda tentang assertion yang dibuat benar atau salah. Jika assertion tersebut salah, maka AssertionError akan terjadi.

Mengaktifkan dan Menontifkan Exceptions

Penggunaan assertions tidak perlu melakukan import package java.util.assert. Menggunakan assertions lebih tepat ditujukan untuk memeriksa parameter dari nonpublic methods jika public methods dapat diakses oleh class lain. Hal itu mungkin terjadi bila penulis dari class lain tidak menyadari bahwa mereka dapat menonaktifkan assertions. Dalam hal ini program tidak dapat bekerja dengan baik. Pada non-public methods, hal tersebut tergunakan secara langsung oleh kode yang ditulis oleh programmer yang memiliki akses terhadap methods tersebut. Sehingga mereka menyadari bahwa saat menjalankannya, assertion harus dalam keadaan aktif.

Untuk mengkompilasi file yang menggunakan assertions, sebuah tambahan parameter perintah diperlukan seperti yang terlihat dibawah ini :

```
javac –source 1.4 MyProgram.java
```

Jika anda ingin untuk menjalankan program tanpa menggunakan fitur assertions, cukup jalankan program secara normal.

```
java MyProgram
```

Namun, jika anda ingin mengaktifkan assertions, anda perlu menggunakan parameter –enableassertions atau –ea.

```
java –enableassertions MyProgram
```

BAB 2
PEMBAHASAN

2.1 Try – Catch

```
class MultipleCatch {
    public static void main(String args[]) {
        try {
            int den = Integer.parseInt(args[0]); //baris 4
            System.out.println(3 / den); //baris 5
        } catch (ArithmeticException exc) {
            System.out.println("Nilai Pembagi  0.");
        } catch (ArrayIndexOutOfBoundsException exc2) {
            System.out.println("Missing argument.");
        }
        System.out.println("After exception.");
    }
}
```

Apakah yang akan terjadi terhadap program bila argumen – argumen berikut dimasukkan oleh user :

a) Tidak ada argument

```
Missing argument.
After exception.
```

Analisis : Program akan mencetak hal tersebut dikarenakan pada saat program berjalan, dia akan mencoba(try) membuat variabel den, yang diisi dengan nilai dari array args(arguments). Akan tetapi nilai pada index yang diminta tidak tersedia, oleh karenanya akan mengambil Exception terhadap ArrayIndexOutOfBoundsException, yang berarti nilai yang diminta tidak berada didalam index array tersebut. Dan program akan mencetak Sysout yang berada dalam ArrayIndexOutOfBoundsException.

b) 1

```
3
After exception.
```

Analisis : Program akan mencetak angka 3 dikarenakan disaat melakukan try yaitu membuat variabel den dengan tipe data int yang diisi nilai array args diindex 0, dan sebelumnya program telah kita beri nilai arguments 1, maka variabel den berisi nilai satu, kemudian melakukan Sysout mencetak hasil penjumlahan 3 / den(1), maka hasilnya 3, maka akan tercetak 3 kelayar, dan dikarenakan tidak memiliki error apapun selama program berjalan, maka tidak ada error yang akan diambil, dan program akan mencetak Sysout terakhirnya dengan lancar.

c) 0

```
Nilai Pembagi  0.
After exception.
```

Analisis : Dapat diliat pada program diatas mencetak Nilai Pembagi 0, yang berarti adalah Sysout dari ArithmeticException, yaitu error yang disebabkan dari hal aritmatika. Error tersebut muncul dikarenakan nilai argumen pada index 0 diisi dengan nilai 0. Oleh karenanya akan terjadi error ketika Sysout 3 / den(0), dan error yang disebabkan berada pada class ArithmeticException, oleh karenanya program akan menampilkan sintak yang ada pada bagian catch ArithmeticException.

2.1.1 Try – Catch Bersarang

```
class NestedTryDemo {
    public static void main(String args[]) {
        try {
            int a = Integer.parseInt(args[0]);
            try {
                int b = Integer.parseInt(args[1]);
                System.out.println(a / b);
            } catch (ArithmeticException e) {
                System.out.println("Divide by zero error!");
            }
        } catch (ArrayIndexOutOfBoundsException ex) {
            System.out.println("2 parameters are required!");
        }
    }
}
```

Apa yang akan terjadi pada program jika argument – argument berikut dimasukkan :

a) Tidak ada argumen

```
2 parameters are required!
```

Analisis : Dikarenakan ketika mengambil nilai array pada variabel args pada index 0 tidak ada, oleh karenanya program mengambil error terhadap ArrayIndexOutOfBoundsException, yang berarti array yang diambil tidak ada.

b) 15

```
2 parameters are required!
```

Analisis : Pada saat pengambilan array args index 0 sudah ada, akan tetapi disaat mencoba mengisi variabel b dengan index 1, index yang diminta tidak ada, oleh karenanya program mengambil error terhadap ArrayIndexOutOfBoundsException. Dan program menampilkan Sysout yang ada pada ArrayIndexOutOfBoundsException.

c) 15 3

```
5
```

Analisis : Pada saat membuat variabel a dan b yang diisi index 0 dan 1, program berjalan dengan lancar, dan program bisa melakukan Sysout terhadap a / b, yang berarti 15 / 3 yaitu hasilnya 5. Dan tidak terjadi error apapun selama program berjalan.

d) 15 0

```
Divide by zero error!
```

Analisis : Pada saat pengisian nilai variabel a dan b terhadap array args yang berada di index 0 dan 1, program berjalan tanpa adanya error, dan error muncul pada saat program melakukan Sysout terhadap a / b, yang berarti 15 / 0, dan disaat pencarian tersebut, program mengalami error dikarenakan nilai tidak boleh dibagi nol, dan error ada di class ArithmeticException, yang berarti program akan mengeksekusi sintak yang ada didalam ArithmeticException tersebut.

Try bersarang lainnya :

```
public class NestedTryDemo2 {
    static void nestedTry(String args[]) {
        try {
            int a = Integer.parseInt(args[0]);
            int b = Integer.parseInt(args[1]);
            System.out.println(a/b);
        } catch (ArithmeticException e) {
            System.out.println("Divide by zero error!");
        }
    }

    public static void main(String args[]){
        try {
            nestedTry(args);
```

```
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("2 parameters are required!");
        }
    }
}
```

Bagaimana output program tersebut jika diimplementasikan terhadap argument – argument berikut :

a) Tidak ada argumen

```
2 parameters are required!
```

Analisis : Pada saat program berjalan, program memanggil method nestedTry dengan mengisi nilai parameter dari array args, dan di method nestedTry, program mencoba mengisi variabel a dan b dengan nilai dari index 0 dan 1 dari array args. Akan tetapi pada array args tidaklah berisi nilai apapun, oleh karenanya program mengembalikan error ArrayIndexOutOfBoundsException, dan mencetak Sysout dari catch tersebut.

b) 15

```
2 parameters are required!
```

Analisis : Sama seperti yang terjadi diatas, akan tetapi pada program ini, pada saat pengisian variabel a dari array args index 0, program berhasil berjalan, akan tetapi disaat pengisian variabel b dengan array args index 1, program mengalami error dikarenakan nilai args pada index 1 tidak ada.

c) 15 3

```
5
```

Analisis : Pada saat memberi nilai variabel a dan b, program akan berjalan tanpa menemui error sedikitpun, oleh karenanya program bisa langsung mencetak hasil dari a / b yang di Sysout di method nestedTry.

d) 15 0

```
Divide by zero error!
```

Analisis : Pada awal program berjalan pada method main, program mencoba memanggil method nestedTry() yang diisi dengan parameter array args. Dan pada method nestedTry(), program mencoba mengisi data variabel a dan b dengan array args yang ada di index 0 dan 1. Dan program mencoba mencetak hasil a / b pada method nestedTry(), akan tetapi terjadi error, dikarenakan program tidak boleh membagi angka dengan 0, oleh karenanya program mengambil error dengan catch ArithmeticException, kemudian program mencetak Sysout dari ArithmeticException tersebut.

2.2 Keyword Finally

Blok finally mengandung kode penanganan setelah penggunaan try dan catch. Blok kode ini selalu tereksekusi walaupun sebuah exception terjadi atau tidak pada blok try. Blok kode tersebut juga akan menghasilkan nilai true meskipun return, continue ataupun break tereksekusi. Terdapat 4 kemungkinan skenario yang berbeda dalam blok try-catch-finally. Pertama, pemaksaan keluar program terjadi bila control program dipaksa untuk melewati blok try menggunakan return, continue ataupun break. Kedua, sebuah penyelesaian normal terjadi jika try-catch-finally tereksekusi secara normal tanpa terjadi error apapun. Ketiga, kode program memiliki spesifikasi tersendiri dalam blok catch terhadap exception yang terjadi. Yang terakhir, kebalikan skenario ketiga. Dalam hal ini, exception yang terjadi tidak terdefinisikan pada blok catch manapun. Contoh dari skenario – skenario tersebut terlihat pada kode berikut ini :

```
public class FinallyDemo {
    static void myMethod(int n) throws Exception{
        try {
            switch(n) {
                case 1: System.out.println("case pertama");
                        return;
                case 3: System.out.println("case ketiga");
                        throw new RuntimeException("demo case
ketiga");
                case 4: System.out.println("case keempat");
                        throw new Exception("demo case keempat");
                case 2: System.out.println("case Kedua");
            }
        } catch (RuntimeException e) {
            System.out.print("RuntimeException terjadi: ");
            System.out.println(e.getMessage());
        } finally {
            System.out.println("try-block entered.");
        }
    }


    public static void main(String args[]){
        for (int i=1; i<=4; i++) {
            try {
                FinallyDemo.myMethod(i);
            } catch (Exception e){
                System.out.print("Exception terjadi: ");
```

```
                System.out.println(e.getMessage());
            }
            System.out.println();
        }
    }

}
```

Analisis :

`for (int i=1; i<=4; i++) {`

➔ Program melakukan looping, dengan variabel i diisi nilai 1, dan mengecek i <= 4, jika benar akan dieksekusi sintaks didalamnya

`try {FinallyDemo.myMethod(i);}`

➔ Program mencoba melakukan pemanggilan method pada class FinallyDemo dengan nama method myMethod() yang diisi dengan parameter nilai variabel i.

`static void myMethod(int n) throws Exception{`

➔ Method yang menerima nilai i dengan parameter n, dan juga method yang akan melempar Exception jika terjadi pelemparan Exception selama pengeksekusian.

```
try {
      switch(n) {
          case 1: System.out.println("case pertama");
                  return;
          case 3: System.out.println("case ketiga");
                  throw new RuntimeException("demo case ketiga");
          case 4: System.out.println("case keempat");
                  throw new Exception("demo case keempat");
          case 2: System.out.println("case Kedua");
      }
}
```

➔ Melakukan perkondisian switch terhadap nilai variabel n yang diisi nilai yang dikirimkan dari method main. Jika nilai n = 1, maka akan mencetak Sysout case 1, dan akan melakukan return, yang berarti akan keluar dari switch, dan finally akan dicetak dikarenakan finally walaupun ada error atau tidaknya, sintak yang ada didalamnya tetap akan tereksekusi. Dan jika nilai n = 2 akan mencetak case kedua. Dan keluar dari switch, dikarenakan tidak ada case yang terletak dibawahnya lagi dan mencetak finally. Dan jika nilai n = 3, akan mencetak case ketiga, dan juga akan melempar

RuntimeException dengan properti demo case ketiga, dan lemparan tersebut akan diterima oleh RuntimeException pada catch, dan mencetak sintaks didalamnya, dan finally tetap akan tereksekusi. Dan jika nilai n = 4, maka akan melakukan case keempat. Dan melempar Exception dengan properti demo case keempat, yang kemudian diterima oleh Exception yang ada di method main, hal ini bisa terjadi dikarenakan pada method myMethod telah diberikan throws yang akan melempar Exception. Kemudian finally akan tetap tereksekusi, dan Exception yang dilempar akan diterima dan kemudian dieksekusi oleh Exception yang ada di method main.

```
catch (RuntimeException e) {
            System.out.print("RuntimeException terjadi: ");
            System.out.println(e.getMessage());
}
```

➔ Berfungsi untuk mengambil RuntimeException pada saat program berjalan terjadi error. Jika ada RuntimeException, maka sintaks didalamnya akan tereksekusi.

```
finally {
            System.out.println("try-block entered.");
}
```

➔ Finally berfungsi untuk tetap akan tereksekusi dalam keadaan program selesai di saat menemukan error ataupun lainnya. Finally sangat berguna jika membuat program yang akan tetap berjalan walau sintaks lainnya terjadi error.

```
catch (Exception e){
                System.out.print("Exception terjadi: ");
                System.out.println(e.getMessage());
}
System.out.println();
```

➔ Jika terjadi Exception, akan tetapi pada program ini, Exception terjadi dikarenakan pelemparan menggunakan throw lewat sintaks disaat program mencapai case keempat, yaitu pelemparan ke Exception, dan method myMethod juga sudah membuat throws yang nanti akan dilempar ke method main. Dan ditangkap oleh Exception pada method main kemudian mengeksekusi sintaks didalamnya.

2.3 Keyword Throw

Disamping menangkap exception, Java juga mengijinkan seorang user untuk melempar sebuah exception. Sintax pelemparan exception cukup sederhana.

```
throw <exception object>;
```

Perhatikan contoh berikut ini.

```
/* Melempar exception jika terjadi kesalahan input */
public class ThrowDemo {
    public static void main(String[] args) {
        String input = "invalid input";
        try {
            if (input.equals("invalid input")) {
                throw new RuntimeException("throw demo");
            } else {
                System.out.println(input);
            }
            System.out.println("After throwing");
        } catch (RuntimeException e) {
            System.out.println("Exception caught here.");
            System.out.println(e);
        }
    }
}
```

Analisis :

```
String input = "invalid input";
```

➔ Memberi nilai invalid input terhadap variabel input yang bertipe data String.

```
try {
    if (input.equals("invalid input")) {
        throw new RuntimeException("throw demo");
    } else {
        System.out.println(input);
    }
    System.out.println("After throwing");
}
```

➔ Mencoba melakukan pengecekan apakah nilai variabel input sama dengan invalid input, jika sama maka akan melakukan pelemparan RuntimeException yang telah diberi properti throw demo dan Sysout Exception caught here dan nilai dari variabel RuntimeException juga akan

dicetak ke layar. Dan jika tidak, maka akan mencetak nilai input beserta Sysout After throwing.

```
catch (RuntimeException e) {
       System.out.println("Exception caught here.");
       System.out.println(e);
}
```

➔ Menangkap RuntimeException yang telah dilempar, kemudian mencetaknya ke layar.

2.4 Keyword Throws

Jika sebuah method dapat menyebabkan sebuah exception namun tidak menangkapnya, maka digunakan keyword throws. Aturan ini hanya berlaku pada checked exception. Anda akan mempelajari lebih lanjut tentang checked exception dan unchecked exception pada bagian selanjutnya, "Kategori Exception".

Berikut ini penulisan syntax menggunakan keyword throws :

```
<type> <methodName> (<parameterList>) throws <exceptionList> {
       <methodBody>
}
```

Sebuah method perlu untuk menangkap ataupun mendaftar seluruh exceptions yang mungkin terjadi, namun hal itu dapat menghilangkan tipe Error, RuntimeException, ataupun subclass-nya.

Contoh berikut ini menunjukkan bahwa method myMethod tidak menangani ClassNotFoundException.

```
class ThrowingClass {
    static void myMethod() throws ClassNotFoundException {
        throw new ClassNotFoundException ("just a demo");
    }
}


public class ThrowsDemo {
    public static void main(String[] args) {
        try {
            ThrowingClass.myMethod();
```

```
        } catch (ClassNotFoundException e) {
            System.out.println(e);
        }
    }

}
```

Analisis :

```
try {
     ThrowingClass.myMethod();
}
```

➔ Mencoba melakukan pemanggilan myMethod yang berada di class ThrowingClass.

```
static void myMethod() throws ClassNotFoundException {
        throw new ClassNotFoundException ("just a demo");
}
```

➔ Membuat method myMethod dengan akses static agar bisa dipanggil oleh method main yang juga memiliki akses static, kemudian membuat throws ClassNotFoundException, yang berfungsi untuk melempar Exception jika terjadi error Exception di class ClassNotFoundException atau pelemparan didalam method tersebut. Kemudian melakukan pelemparan ClassNotFoundException beserta properti just a demo.

```
catch (ClassNotFoundException e) {
       System.out.println(e);
}
```

➔ Berfungsi mengambil ClassNotFoundException jika terjadi pelemparan dari method yang telah dipanggil.

catch lebih dari satu harus berurutan dari subclass ke superclass.

```
8   public class MultipleCatchError {
9       public static void main(String[] args) {
10          try {
11              int a = Integer.parseInt(args[0]);
12              int b = Integer.parseInt(args[1]);
13              System.out.println(a/b);
            } catch (Exception e) {
15              System.out.println(e);
            } catch (ArrayIndexOutOfBoundsException e2) {
17              System.out.println(e2);
18          }
19          System.out.println("After try-catch-catch.");
20      }
21
22  }
```

Setelah mengkompilasi kode tersebut akan menghasilkan pesan error jika Exception class adalah superclass dari ArrayIndexOutOfBoundsException class.

```
MultipleCatchError.java:9: exception
java.lang.ArrayIndexOutOfBoundsException has already been
caught  } catch (ArrayIndexOutOfBoundsException e2) {
```

Analisis :

Program diatas terjadi error dikarenakan Exception dari ArrayIndexOutOfBoundsException, telah diambil oleh Exception superclass, oleh karenanya pengambilan Exception oleh subclass akan mengakibatkan error.

2.5 User Defined Exceptions

Meskipun beberapa exception classes terdapat pada package java.lang namun tidak mencukupi untuk menampung seluruh kemungkinan tipe exception yang mungkin terjadi. Sehingga sangat mungkin bahwa anda perlu untuk membuat tipe exception tersendiri.

Dalam pembuatan tipe exception anda sendiri, anda hanya perlu untuk membuat sebuah extended class terhadap RuntimeException class, maupun Exception class lain. Selanjutnya tergantung pada anda dalam memodifikasi class sesuai permasalahan yang akan diselesaikan. Members dan constructors dapat dimasukkan pada exception class milik anda.

Berikut ini contohnya :

```
class HateStringException extends RuntimeException{
    /* Tidak perlu memasukkan member ataupun construktor */
}

public class TestHateString {
    public static void main(String[] args) {
        String input = "invalid input";
        try {
            if (input.equals("invalid input")) {
                throw new HateStringException();
            }
            System.out.println("String accepted.");
        } catch (HateStringException e) {
            System.out.println("I hate this string: " + input +
".");
        }
    }
}
```

Analisis :

```
String input = "invalid input";
```

➔ Mendeklarasi variabel input dengan tipe data String dan bernilai “invalid input”

```
try {
    if (input.equals("invalid input")) {
        throw new HateStringException();
}
System.out.println("String accepted.");
```

➔ Mencoba melakukan pengecekan kondisi yaitu jika nilai input sama dengan “invalid input”, maka akan dilakukan pelemparan method yang memiliki Exception yang telah dideklarasi. Dan jika input tidak sama dengan “invalid input”, maka akan melakukan Sysout String accepted.

```
class HateStringException extends RuntimeException{
    /* Tidak perlu memasukkan member ataupun construktor */
}
```

➔ Membuat method yang menerima pewarisan dari RuntimeException, dengan begitu, method ini bisa menjadi Exception yang dibuat oleh user yang memiliki pewarisan dari RuntimeException.

```
catch (HateStringException e) {
      System.out.println("I hate this string: " + input + ".");
}
```

➔ Melakukan penangkapan jika terjadi pelemparan HateStringException yang telah dideklarasi oleh user menggunakan method yang menerima pewarisan dari RuntimeException. Kemudian melakukan Sysout I hate this string beserta nilai variabel input.

2.6 Assertions

Penulisan assertions memiliki dua bentuk.
Bentuk yang paling sederhana terlihat sebagai berikut :

```
assert <expression1>;
```

dimana *<expression1>* adalah kondisi dimana assertion bernilai true.

Bentuk yang lain menggunakan dua ekspresi, berikut ini cara penulisannya :

```
assert <expression1>: <expression2>;
```

dimana *<expression1>* adalah kondisi assertion bernilai true dan *<expression2>* adalah informasi yang membantu pemeriksaan mengapa program mengalami kesalahan.

```
public class AgeAssert {
    public static void main(String[] args) {
        int age = 19;
        assert(age>0);
        /* jika masukan umur benar (misal, age>0) */
        if (age >= 18) {
            System.out.println("Congrats! You're an adult! = ");
        }
        System.out.println("Selamat");
    }
}
```

Hasil ketika age = 19 :

```
Congrats! You're an adult! =
Selamat
```

Analisis :

Program diatas akan berjalan lancar dikarenakan nilai yang dimasukkan sudah memenuhi atau memperoleh true dari asumsi yang diberikan.

Hasil ketika age = 0 :

```
Exception in thread "main" java.lang.AssertionError
        at Belajar1.AgeAssert.main(AgeAssert.java:10)
C:\Users\LENOVO\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

Analisis :

Program mengalami error dan pemberhentian paksa, dikarenakan nilai yang ditampung pada variabel age tidak memenuhi asumsi yang diberikan, yaitu age>0. Oleh karenanya program akan menampilkan error dan akan berhenti secara otomatis.

BAB 3
PENUTUP

3.1 Kesimpulan

Exception adalah suatu penanganan terhadap error yang terjadi disaat program sedang berjalan. Exception berguna untuk menangani error yang terjadi, kemudian mengambil error tersebut dan mengeksekusi untuk menampilkan pesan error yang terjadi atau pesan yang dibuat oleh user pembuat program. Dalam pembuatan Exception, akan selalu diikuti oleh keyword try-catch. Keyword tersebut berguna untuk mencoba melakukan sesuatu didalam try, dan jika ditemui error maka catch akan mengambil error tersebut menggunakan Exception atau class dari error yang terjadi.

Exception menyediakan begitu banyak class yang bisa digunakan, dan class-class tersebut mendefinisikan dari error yang terjadi. Dan adapun user juga bisa membuat Exception punya sendiri dengan cara mendefinisikan method yang memiliki nilai bawaan dari suatu class Exception yang ada.

Adapun keyword throw dan throws akan sangat berguna selama kita membuat suatu Exception yang akan menangani error yang terjadi. Perbedaan keyword throw dan throws adalah :

1. Throw digunakan untuk melakukan pelemparan secara explicit ke Exception.
2. Throw dapat digunakan untuk Exception checked dan unchecked Exception.
3. Throw biasanya digunakan untuk melempar Exception yang custom.
4. Throws digunakan untuk mendeklarasi jika terjadi Exception pada method yang telah dibuat throws, maka method tersebut akan melempar Exception tersebut[2].

Adapun keyword assert digunakan untuk membuat suatu asumsi dengan expresi1 harus diisi dengan nilai yang akan menghasilkan true atau false, dan jika nilai yang dihasilkan false, maka program akan berhenti dan menampilkan error yang terjadi, dan jika true, program akan berjalan dengan lancar[3].

DAFTAR PUSTAKA

[1] Thamura, Frans. *Modul JENI-intro2: Bab02 Exceptions dan Assertions.* JENI.

[2] Rini Wongso, S.Kom., M.T.I. Throw vs Throws pada Exception Handling. https://socs.binus.ac.id/. Diakses pada 05 Maret 2021.

[3] Viska Mutiawani dan Kurnia Saputra. 20 Maret 2015. Assertion dan Exception Handling. http://informatika.unsyiah.ac.id/. Informatika Unsyiah. Diakses pada 05 Maret 2021.